INDUCCIÓN DE MACRO-OPERADORES DE REPARACIÓN EN SISTEMAS INTERACTIVOS DE RE-SCHEDULING

Paula A. Toselli[1], Jorge A. Palombarini[2], Ernesto C. Martínez[3]

[1]*Departamento de Sistemas, (UTN-FRVM)*, Av. Universidad 450, Villa María 5900, Argentina.
[2] *GISIQ (UTN-FRVM)*, Av. Universidad 450, Villa María 5900, Argentina.
[3]*INGAR (CONICET-UTN)*, Avellaneda 3657, Santa Fe, S3002 GJC, Argentina.

Con la tendencia actual hacia los sistemas cognitivos de manufactura para tratar con eventos imprevistos y perturbaciones que constantemente demandan una respuesta rápida a eventos disruptivos, la integración de capacidades de aprendizaje/razonamiento e interactividad es fundamental para la reprogramación en tiempo real de cronogramas de tareas en el piso de planta, teniendo en cuenta distintos objetivos y metas de reparación. El presente trabajo propone un mecanismo para la generación, actualización e incorporación automática de conocimiento experto en la reparación de cronogramas de tareas a través de interacción con un sistema de aprendizaje utilizando transiciones simuladas de estados abstractos de un *schedule*. Representaciones déicticas de los cronogramas de tareas basadas en puntos focales son utilizadas para inducir secuencias de operadores de reparación (macro-operadores) guiadas por objetivos para enfrentar eventos no planificados y perturbaciones operacionales. Un ejemplo industrial donde se necesita reprogramar tareas debido al arribo de una nueva orden es discutido utilizando el prototipo SmartGantt para realizar la re-planificación interactiva en tiempo real y la integración automática en el sistema del conocimiento inducido a partir de la interacción entre el experto y el sistema.

Introducción

La tendencia emergente hacia los sistemas cognitivos de planificación y control de la producción, ha producido que la introducción de mecanismos de aprendizaje y la generación/ actualización en-línea de conocimiento para *re-scheduling* utilizando reglas heurísticas que puedan ser aplicadas en tiempo real tiene una importancia crítica en cualquier estrategia de gestión de disrupciones, para garantizar el cumplimiento de *due dates* de órdenes en curso [1]. En este contexto, son esenciales atributos tales como reactividad y rapidez de respuesta del sistema de producción, por lo cual es clave generar *schedules* satisfactorios en lugar de óptimos, en un tiempo de computación razonable [2]; no obstante, el *re-scheduling* puramente automático no es realista, debido a que no contempla el importante rol del experto humano, quien posee la responsabilidad final de todas las decisiones y debe ser involucrado en el proceso de solución ([2],[3]).

Por consiguiente, las mencionadas circunstancias han generado la necesidad de diseñar e implementar mecanismos que proporcionen al sistema de *re-scheduling* la capacidad de integrar en sus políticas de reparación aprendidas vía simulación intensiva, tácticas inducidas a partir de reparaciones realizadas interactivamente por operadores humanos expertos, quienes son capaces de reprogramar una gran cantidad de tareas y recursos aprendiendo de manera incremental una estrategia de reparación que utiliza como abstracción un número de objetos (tareas, recursos) con atributos y relaciones (precedencia, sincronización) entre ellos [4]. De esa manera, en el presente trabajo se propone un mecanismo de inducción automática vía aprendizaje inverso, de macro-operadores de reparación que pueden ser utilizados en estrategias de *re-scheduling* en tiempo real. Basados en representaciones déicticas de los estados del schedule ([4],[5]), se han empleado micro-operadores locales que permiten al operador humano efectuar acciones correctivas sobre el *schedule* actual. La inducción de conocimiento de control sobre la secuencia de aplicación de dichos operadores, es realizada a través de un algoritmo de *Inverse Reinforcement Learning* (IRL) ([6],[7]), utilizando un simulador de estados del schedule, en el cuál una instancia del mismo se modifica interactivamente, hasta arribar a un estado objetivo o meta de reparación. El conocimiento adquirido es almacenado como un nuevo macro-operador disponible para futuras reparaciones automáticas, cuyo valor de aplicación se actualiza utilizando un algoritmo de *Reinforcement Learning* ([8]).

Una implementación prototipo del enfoque propuesto es presentada, utilizando Visual Basic.NET 2008 y Matlab R2011. Como ejemplo representativo, el prototipo es utilizado para inducir conocimiento de *re-scheduling* en una planta batch; dicho conocimiento, se integra en

políticas aprendidas vía simulación, demostrando las ventajas de su incorporación a la política de reparacion, reflejadas en mejoras sustanciales en la performance del agente de re-scheduling.

Re-scheduling basado en reparación

La inducción automática de macro-operadores de reparación presentada en el presente trabajo se ha desarrollado como característica adicional en el marco de una arquitectura basada en reparación que se muestra en la Fig. 1, implementada por el prototipo SmartGantt [4] y embebida en una estructura más general que incluye una función ERP *(Enterprise Resource Planning System)*) y un sistema de control de ejecución MES (*Manufacturing Execution System*) con una infraestructura que incluye capacidades de comunicación y control, e integra capacidades cognitivas artificiales en recursos y procesos para incluir por diseño flexibilidad y adaptabilidad en los sistemas de producción [9]. En este enfoque, el conocimiento para rescheduling en tiempo real utilizando una secuencia óptima de operadores de reparación se genera de dos maneras distintas: a través de refuerzos y micro-operadores utilizando un simulador de estados del schedule, o a través de interacción con el experto humano (induciendo macro-operadores en modo *expert*), actualizando ambas la misma base de conocimiento para su posterior consulta automática.

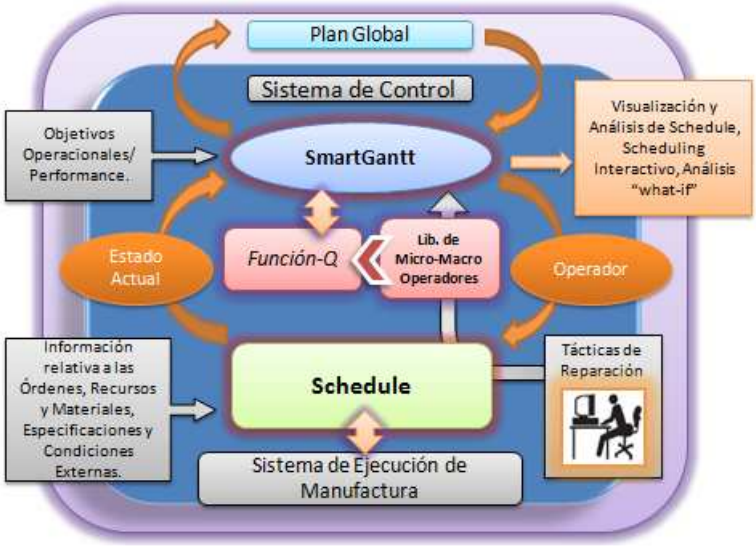

Figura 1. Arquitectura Basada en Reparación

En el modo *expert*, una instancia del schedule es modificada interactivamente, a través de la ejecución de acciones correctivas utilizando una secuencia de operadores de reparación hasta que un determinado objetivo se alcanza. En cada episodio de aprendizaje, SmartGantt recibe información del estado actual *s* del schedule (representado a través de un vector de *features*), y entonces proporciona una lista de operadores de reparación al experto humano. Posteriormente, el operador seleccionado es aplicado al schedule actual, resultando en nuevo estado del schedule, hasta lograr al estado objetivo. A continuación, empleando un algoritmo de IRL el sistema aprende la función de *reward* utilizada por el experto en la reparación interactiva, y almacena el conocimiento adquirido en su Librería de Micro/macro-operadores. En modo simulación, el proceso es similar, excepto por el hecho de que la reparación se realiza de manera automática por el sistema, utilizando los macro/micro-operadores disponibles.

```
for each s ∈S and a∈A do
      initialize table entre Q(s, a)
end for
generate a starting state s
repeat
      select an action a and execute it
      receive an inmediate reward r=r(s, a)
      observe the new state s'
      update the table entry for Q(s, a) as follows:
      Q(s,a) ← r + γ max_b Q(s',b)
      s← s'
until no more learning episodes
```

Figura 2. Algoritmo *Q-Learning* Básico

La evaluación de la calidad resultante del schedule después de que un dado operador de reparación ha sido aplicado se realiza en SmartGantt a través de una función de *reward r(s)*. El sistema de aprendizaje actualiza entonces su función de acción-valor *Q(s,a)* que estima el valor o utilidad de aplicar el macro/micro-operador de reparación en un estado del schedule *s* dado. Tal actualización es realizada utilizando un algoritmo de *Reinforcement Learning* (RL) [8] como *Q-Learning*, el cual se muestra en la Fig. 2. Acumulando suficiente experiencia a partir de muchas transiciones simuladas, SmartGantt es capaz de aprender una política óptima para seleccionar el mejor macro/micro-operador de reparación en cada estado del schedule.

El principal beneficio de aplicar técnicas de RL tales como el algortimo *Q-Learning* para la búsqueda de conocimiento de control para la mejora de la calidad y la eficiencia del *re-scheduling* en tiempo real es que no se depende de la disponibilidad de expertos en el dominio (aunque permite incorporar su conocimiento, como se muestra en el presente trabajo), la adaptación online a un entorno dinámico y la posibilidad de utilizar abstracciones que son necesarias para tratar con espacios de estados extensos (e.g. cadenas de suministro) [4]. Para reparar un schedule, SmartGantt utiliza una función objetivo $G:S \rightarrow \{true,false\}$ la cual define cuáles estados en el schedule reparado definen la meta de la reparación, e.g. estados donde la tardanza total es menor o igual a 1 día de trabajo. Usualmente, una función de precondición como *pre*: $S \times A \rightarrow \{true, false\}$ es utilizada para especificar cuál subconjunto de operadores de reparación puede ser aplicado en cada estado posible del schedule para tener en cuenta capacidades del recurso y restricciones de precedencia (e.g., recetas de producto) así como también acciones de reparación alternativas tales como dividir una tarea, unir dos tareas, intercambiar tareas entre recursos, etc.

El objetivo de una tarea de reparación puede establecerse como: "dado un estado inicial *s1* para el schedule, encontrar una secuencia de operadores de reparación $a_1, a_2,\ldots,a_n$ *con* $a_i \in A$ tal que $goal(\delta(\ldots\delta(s_1,a_1)\ldots,a_n)) = true$ donde δ es la función de transición, la cuál es únicamente revelada al agente aprendiz a través de simulación." La función de *reward* mencionada se utiliza para traducir el objetivo en una señal de refuerzo para guiar la búsqueda de una política que seleccione el mejor operador de reparación en cada uno de los estados posibles del schedule.

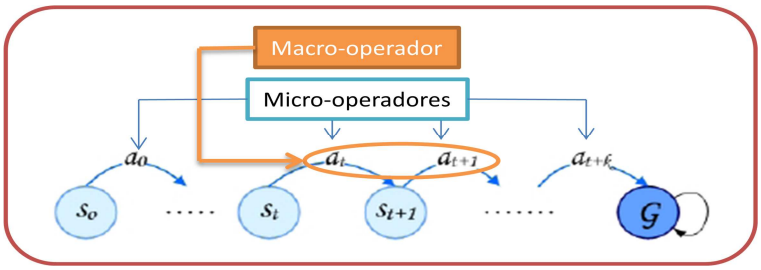

Figura 3. Micro/macro-operadores

Como puede observarse en la Fig. 3, los macro-operadores constituyen secuencias finitas de micro-operadores que pueden aplicarse de manera independiente o combinada con otros micro/macro-operadores. Basándose en la función de *reward* y simulaciones de transición de estados, la política óptima $a_i = \pi^*(s_i)$ puede ser aproximada utilizando el algoritmo Hedger en combinación con Locally Weighted Regression [10], y luego ser empleada para computar la secuencia de acciones que permite alcanzar un schedule reparado, mejorando la capacidad de respuesta en el piso de planta para manipular eventos no planificados y gestionar perturbaciones [4]. A su vez, diferentes objetivos operacionales y de performance, así como también preferencias del usuario, y tácticas de reparación *especializadas* pueden ser provistas a SmartGantt por el experto humano. Se proveen además características de visualización, soluciones alternativas y capacidades de análisis de forma que el usuario pueda decidir en forma totalmente interactiva utilizando una interfaz gráfica que le permite alcanzar objetivos de *re-scheduling* alternativos.

Inverse Reinforcement Learning

Los métodos de *Inverse Reinforcement Learning* (IRL) [6] aprenden una función de *reward* en un Proceso Decisorio Markoviano (PDM) a partir de demostraciones de los expertos, permitiendo la generalización de la política de los mismos a situaciones nunca vistas. El desafío central en IRL es encontrar una función de *reward* con una estructura significativa que represente la tarea de manera compacta y portable. En el presente trabajo se ha utilizado el algoritmo *Gaussian Process Inverse Reinforcement Learning* (GPIRL) [7], que extiende el modelo de *Gaussian Processes* (GP) para aprender funciones de *reward* altamente no lineales que de manera consistente capturen el

comportamiento demostrado por el experto. El algoritmo GPIRL únicamente necesita observar las acciones del experto, y no asume que el mismo actúa necesariamente de manera óptima. Los hiperparámetros del kernel en GPs aprendidos capturan la estructura del *reward* incluyendo la relevancia de cada componente del estado. Una vez aprendido, el GP puede ajustar el *reward* para el espacio de estados, y proporcionar predicciones para estado no visitados, combinando razonamiento probabilístico acerca del comportamiento estocástico del experto con la habilidad de aprender el *reward* como una función no lineal utilizando ejemplos posiblemente sub-óptimos, como es el caso en el *re-scheduling* de tareas. De esa manera, el GP provee un método para aprender los hiper-parámetros del mencionado kernel así como también la estructura de la función (desconocida) de *reward*. Al efecto se ha utilizado:

$$logP(\mathrm{D}|\mathrm{r}) = \sum_i \sum_t \log P\left(a_{i,t}|s_{i,t}\right) = \sum_i \sum_t (Q^r_{\mathrm{s(i,t)a(i,t)}} - V^r_{s_{i,t}}) \quad (1)$$

para especificar una distribución sobre las salidas del GP y aprender los valores de salida $\boldsymbol{u}$ y la función kernel para predecir los *rewards* asociados con el vector de *features* $\boldsymbol{X_u}$. Los *rewards* para estados no incluidos en los datos de entrenamiento son inferidos por el GP. Los hiperparámetros $\boldsymbol{\theta}$ del kernel son aprendidos para modelar la estructura del *reward*. Los valores más probables de $\boldsymbol{u}$ y $\boldsymbol{\theta}$ se encuentran maximizando su probabilidad a partir del conjunto *D* de demostraciones del experto (Ec. 2):

$$P(u,\theta|D,X_u) \propto P(D,u,\theta|X_u) = \left[\int_r \underbrace{P(D|r)}_{IRL}\ \underbrace{P(r|u,\theta,X_u)}_{GP\ posterior}\ dr\right] \underbrace{P(u,\theta|X_u)}_{probabilidad\ GP} \quad (2)$$

El *log* de *P(D|r)* se obtiene mediante la Ec. (1), el GP posterior $P(r|u;\theta;X_u)$ es la probabilidad de una determinada función de *reward* a partir de los valores actuales de $\boldsymbol{u}$ y $\boldsymbol{\theta}$, y $P(u;\ \theta|X_u)$ es la probabilidad previa de una particular asignación de $\boldsymbol{u}$ y $\boldsymbol{\theta}$. El *log* de $P(u;\theta|X_u)$ es el log de probabilidad marginal del GP, el cual favorece el uso de funciones de kernel simples y valores de $\boldsymbol{u}$ que se ajusten a la matriz kernel actual (Ec. 3):

$$\log P(u,\theta|X_u) = -\frac{1}{2}u^T K_{u,u}^{-1}\, u - \frac{1}{2}\log\left|K_{u,u}\right| - \frac{n}{2}\log 2\pi + \log P(\theta) \quad (3)$$

El último término l*og P(θ)* es un hiperparámetro, y las entradas de la matriz de covarianza $\boldsymbol{K_{u;u}}$ son establecidas por la función kernel. Para determinar la relevancia de cada *feature* se ha utilizado un kernel de Detección de Relevancia Automática (DRA) con hiperparámetros θ ={B,Δ} (Ec. 4):

$$k(x_i,x_j) = \beta\ exp\left(-\frac{1}{2}(x_i - x_j)^T \Lambda\,(x_i - x_j)\right) \quad (4)$$

El hiperparámetro B es la varianza total, y la matriz diagonal Δ especifica el peso de cada *feature*. Cuando Δ es aprendida, los *features* menos relevantes reciben pesos bajos, y los más relevantes reciben pesos altos.

Caso de Estudio Industrial

La aplicación prototipo SmartGantt utilizada como base para el desarrollo realizado en el presente trabajo ha sido implementada en Visual Basic .NET 2008 Development Framework 3.5 y Matlab R2011 corriendo bajo Windows 7, y se describe en detalle en [4]. Además, se ha empleado el *toolkit* IRL descripto en [7]

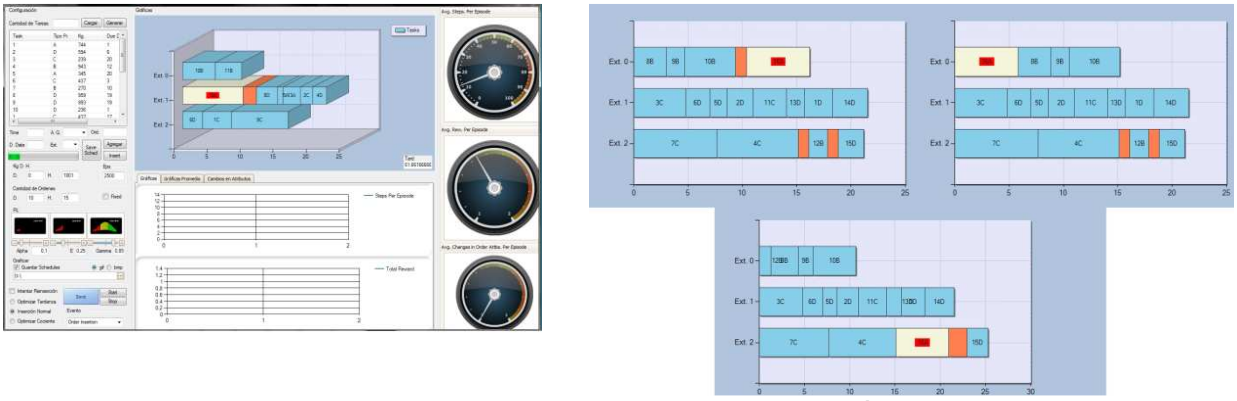

Figura 4. Pantalla del prototipo (izquierda) y una demostración del experto (derecha)

La Fig. 4 muestra la interfaz gráfica de usuario para la aplicación desarrollada, así como también uno de los 8 ejemplos de reparación interactiva proporcionados por el experto. Originalmente el prototipo admite dos modos de uso: *training* y *consult*. Durante el modo *training* SmartGantt aprende a reparar *schedules* a través de transiciones simuladas, y el conocimiento generado es codificado en la función *Q*. La explotación del conocimiento de *re-scheduling* es realizada en el modo *consult*. Adicionalmente, para el presente trabajo se ha incorporado un tercer modo de operación denominado *expert*, en el cual el sistema presenta de manera sucesiva al operador humano una serie de *schedules* que deben ser reparados interactivamente ante el arribo de una nueva orden que necesita ser insertada sin incrementar la tardanza presente en el sistema. Una vez finalizado el proceso de reparación, SmartGantt aplica el algoritmo GPIRL a través de una interfaz COM con Matlab. Para implementar la funcionalidad necesaria se ha creado una variable tipo "struct" que contiene los ejemplos a correr en el algoritmo en forma de celdas estado/acción, y otra estructura interna donde se definen los parámetros del PDM. Para correr los ejemplos de entrenamiento se ha empleado el módulo "Human Control/runtest", que permite ejecutar un script con el entrenamiento deseado estableciendo los parámetros correspondientes para cada modelo. Para el presente trabajo se ha utilizado un 'standardmdp' parametrizado con las variables mencionadas. Los resultados arrojados por el script han sido "parseados" utilizando las librerías ExtremeOptimization.NET[12], y agregados de manera dinámica a la Librería de operadores para ser utilizados luego en modo *training*.

Un ejemplo de problema propuesto en [11] se ha considerado para ilustrar el uso del prototipo en modo *expert* para adquirir conocimiento de *re-scheduling* en una planta batch. La misma está compuesta por 3 extrusoras semi-continuas que procesan órdenes de clientes para 4 tipos de productos. Cada extrusora posee características distintivas, de manera tal que no todas pueden procesar todos los productos. Adicionalmente, los rates de procesamiento dependen tanto del recurso como del producto que está siendo procesado, y para mayor detalle en la simulación en base a las relaciones de precedencia se han introducido tiempos de set-up destinados a la limpieza de los recursos [4]. El estado del schedule se ha caracterizado a través de un vector de 7 *features* locales relacionados con la tarea que se toma como foco de aplicación de los operadores, y 10 *features* globales relacionados con la totalidad del schedule [5]

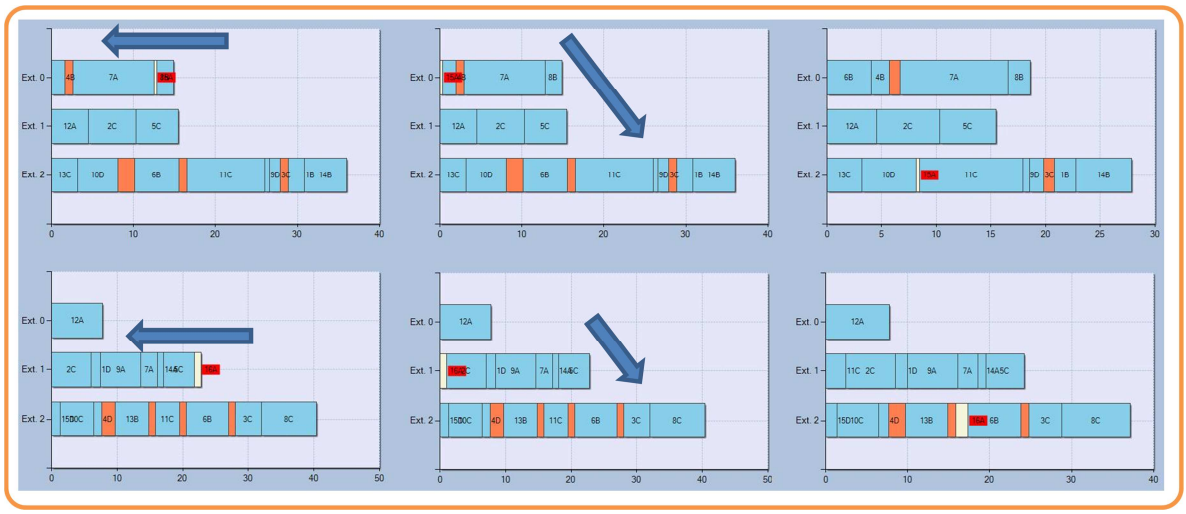

Figura 5. Dos ejemplos de la aplicación en modo *training* del macro-operador *LeftJump-DownRightSwap* inducido a partir de la interacción con el experto.

En la Fig. 5 pueden observarse los beneficios de la aplicación del macro-operador *LeftJump-DownRightSwap* aprendido, que le permite al sistema calcular el valor *Q* de la utilización de la secuencia completa como si se tratara de un solo operador. La figura demuestra que el macro-operador inducido puede ser aplicado en situaciones no necesariamente similares. Por caso, en la sección superior la tarea focal se encuentra en el primer recurso, en la inferior en el segundo, la cantidad total de tareas y la tardanza es distinta, etc. En el primer caso la aplicación del macro-operador ha logrado reducir la tardanza total en 3.23 h. (TI=11.50 h. y TF=8.27 h.) y en el segundo 1.39 h. (TI=13.12 h. y TF=11.73 h.) en ambos casos incluyendo la nueva orden.

Conclusión

Se ha presentado un enfoque novedoso para la inducción interactiva de conocimiento de *re-scheduling* en la forma de macro-operadores y su implementación en la aplicación prototipo

SmartGantt. Dicho conocimiento se ha derivado utilizando *Inverse Reinforcement Learning* a partir de reparaciones realizadas por expertos humanos, y luego incorporado de manera online a una librería de operadores para ser empleado posteriormente en el aprendizaje vía simulación intensiva de políticas de reparación automática de *schedules* en tiempo real, por medio de la integración del *Reinforcement Learning* con *Locally Weighted Regression*. Las políticas aprendidas permiten la generación de secuencias de macro/micro-operadores de reparación deícticos locales para alcanzar objetivos de *re-scheduling* y de esa manera gestionar eventos anormales no previstos tales como la inserción de una nueva orden que arriba al sistema sin incrementar de la tardanza presente en el mismo. Si bien como trabajo futuro se ha planteado al análisis del impacto cuantitativo en el proceso de convergencia del algoritmo de aprendizaje a través de la incorporación en las secuencias de reparación de macro-operadores inducidos, se ha mostrado un ejemplo de aplicación en el cuál el sistema SmartGantt es expuesto a una serie de reparaciones realizadas por un experto humano, a partir de las cuales debe generar un macro-operador que responde a la táctica utilizada por el mismo durante el proceso interactivo de reparación. Finalmente, el macro-operador aprendido ha sido integrado en la política de reparación resultante de la simulación intensiva, demostrándose los importantes beneficios obtenidos, los que quedan reflejados tanto a partir de la reducción de tardanza presente en el sistema como en la portabilidad del conocimiento adquirido. Esto representa una táctica general aplicable en tiempo real de manera automática en schedules no necesariamente similares.

Referencias

[1] Vieira, G., Herrmann, J., Lin, E. Rescheduling Manufacturing Systems: a Framework of Strategies, Policies and Methods. *J. of Scheduling*, 6, 39 (2003) 39–62.

[2] Henning, G., Cerdá, J. Knowledge-based predictive and reactive scheduling in industrial environments. *Computers and Chemical Engineering* 24 (2000) 2315–2338.

[3] Aytug, H., Lawley, M., McKay, K., Mohan, S., Uzsoy, R. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161, (2005) 86‑110.

[4] Palombarini, J., Martinez, E. SmartGantt ‑ An intelligent system for real time rescheduling based on relational reinforcement learning. *Expert Systems with Applications* 39 (2012) 10251-10268.

[5] Miyashita, K. Learning scheduling control knowledge through reinforcements. *International Transactions in Operational Research*, 7(2), (2000) 125-138.

[6] Abeel, P., Ng, A. Apprenticeship Learning via Inverse Reinforcement Learning. *En Proc. 21st International Conference on Machine Learning*, ACM Press, NY (2004).

[7] Levine, S., Popovic, Z., Koltun, V. Nonlinear Inverse Reinforcement Learning with Gaussian Processes. *Advances in Neural Information Processing Systems 24,* (2011).

[8] Sutton, R., Barto, A. Reinforcement Learning: An Introduction. MIT Press, Boston, (1998)

[9] Trentesaux, D. Distributed control of production systems. *Engineering Applications of Artificial Intelligence*, 22, (2009) 971–978.

[10] Smart, W., Kaelbling, L., Practical Reinforcement Learning in Continuous Spaces. *En Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc. (2000), 903-910.

[11] Musier, R., Evans, L. An approximate method for the production scheduling of industrial batch processes with parallel units. *Computers and Chemical Engineering*, 13, (1989) 229‑238.

[12] Extreme Optimization Numerical Libraries for .NET. http://www.extremeoptimization.com/. 29/03/2012